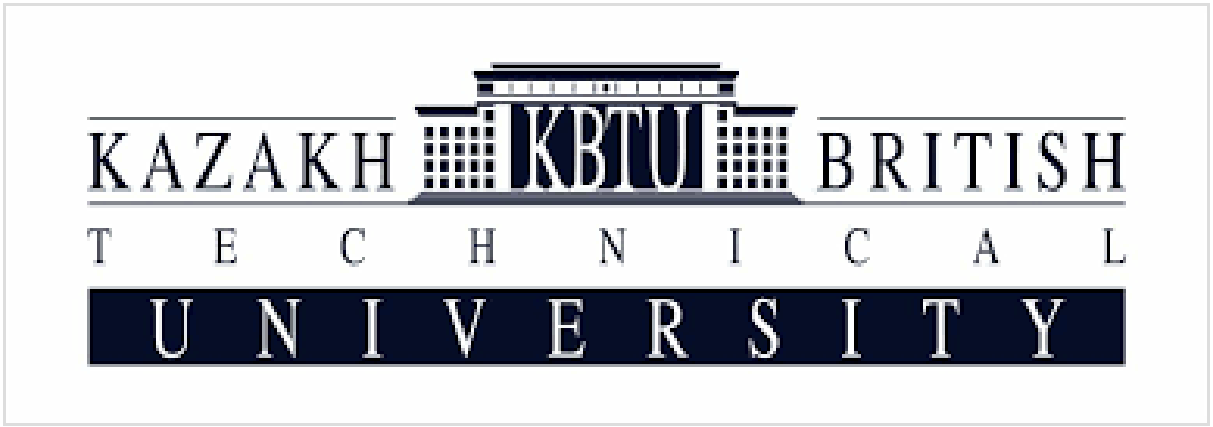


Project name: University system
Discipline: Object-Oriented programming

Team members**:**
Beisenbayeva Diana
Mubarakuly Magzhan
Ybyraiym Aktanberdi
Seitova Madina

Main Goal of the Project

Create an Intranet-based University Management System that streamlines the distribution, coordination, and implementation of key administrative tasks within an educational institution. The system should facilitate high-quality interactions between university employees and students by providing convenient, basic queries through a console-based Java application. This project leverages core object-oriented programming concepts learned in coursework to model and manage the various components and relationships within the university environment.

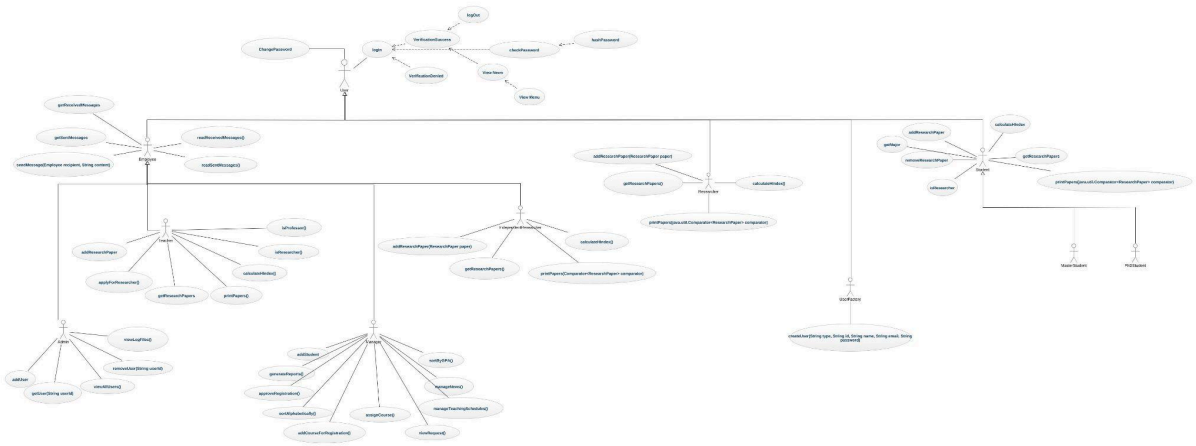
Project description:

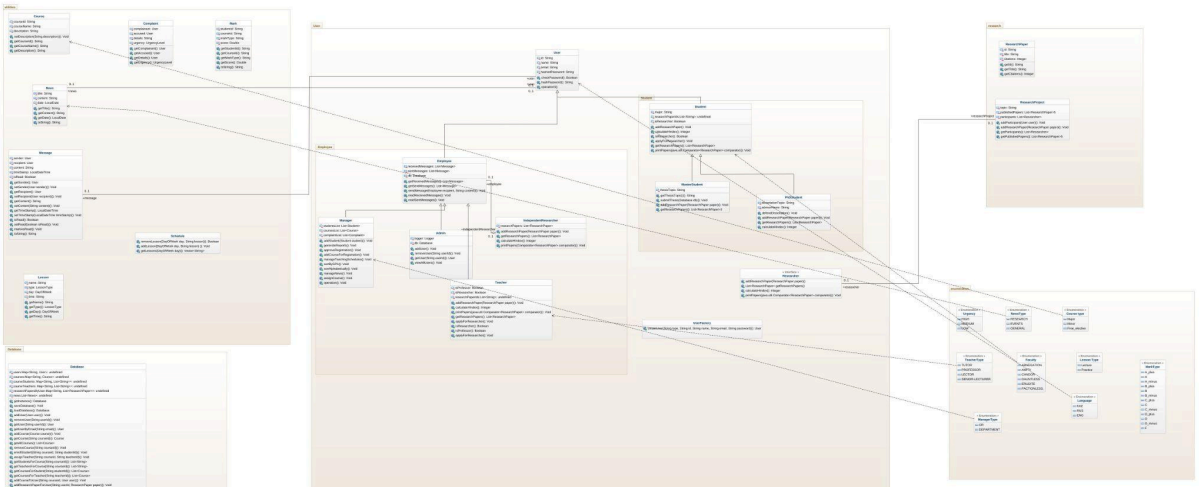
1. Create an architecture using Use Case and UML class Diagrams.
2. Working with Eclipse
3. Report, documentation, presentation

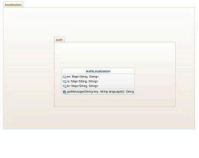
1. Diagrams:

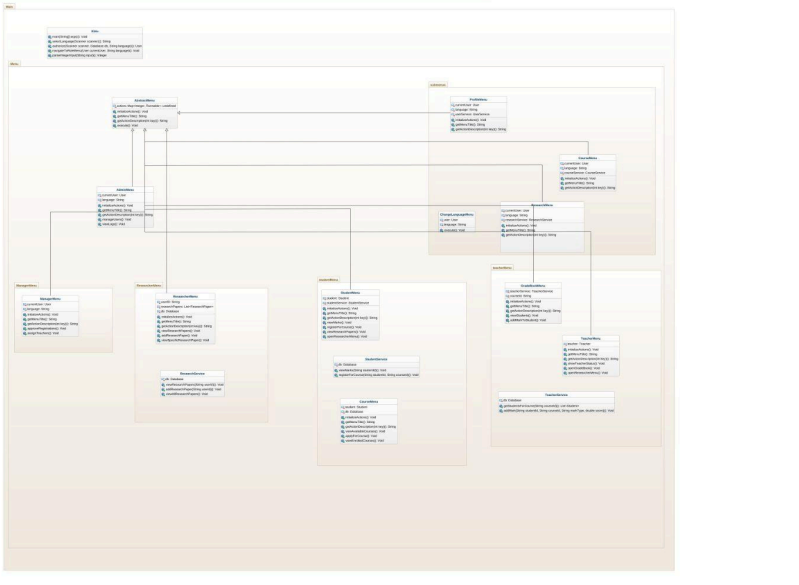
We have created diagrams using GenMyModel UML tool. The UML divides into two parts:

- Use-case diagram
- Class diagram


logOut
ChangePassword
login
VerificationSuccess
checkPassword
hashPassword
VerificationDenied
View News
View Works
Use
getReceivedMessages
getSentMessages
readReceivedMessages()
sendMessage(Employee recipient, String content)
readSentMessages()
Employee
isProfessor()
isResearcher()
addResearchPaper
calculateHIndex()
applyForResearcher()
getResearchPapers
printPapers()
Teacher
addResearchPaper(ResearchPaper paper)
getResearchPapers()
printPapers(Comparator<ResearchPaper> comparator)
calculateHIndex()
addResourcePaper(ResourcePaper paper)
getResourcePapers()
calculateHIndex()
printPapers(Comparator<ResourcePaper> comparator)
ResearchAce
addResearchPaper
getMajor
removeResearchPaper
calculateHIndex()
getResourcePapers()
isResearcher
printPapers(java.util.Comparator<ResearchPaper> comparator)
Master.Student
PhD.Student
Admin
viewLogFiles()
removeUser(String userId)
addUser
getUser(String userId)
viewAllUsers()
Librarian
addStudent
notify(GPA)
generateReports()
signageNews()
approveRegistration()
sortAlphabetically()
assignCourse()
manageTeachingSchedules()
addCourseForRegistration()
viewRequest()
StorFacade
createUser(String type, String id, String name, String email, String password)







2. Working with Eclipse

Packages

We divided our project into 6 packages

- database (package where our data is stored: Database)
- localization (resources for localization (multi-language support))
- main (menus and database)
- research (contains researcher paper and project)
- users (employees, students, and researchers)
- utilities (helper classes and methods used across various parts of the application)

The `User` class serves as an abstract base class for all users in the system, such as students, teachers, and administrators. It implements the `Serializable` interface to allow instances of the class to be serialized.

```
1   package users;
2
3   import java.io.Serializable;
4   import java.util.Objects;
5
6   public abstract class User implements Serializable {
7       private final String id;
8       private final String name;
9       private final String email;
10      private final String hashedPassword;
11
```

The `Student` class extends the `User` class and implements the `Researcher` interface, enabling students to participate in research projects and manage academic activities.

```
public class Student extends User implements Researcher {
    private String major;
    private boolean isResearcher;
    private final List<ResearchPaper> researchPapers;
    private Schedule schedule;
    private final List<Course> registeredCourses;
    private final List<Mark> marks;
    private final Set<Organizations> joinedOrganizations;
    private final Map<String, Integer> retakeCount;
```

The system provides robust schedule management functionalities, allowing users to add, remove, and view lessons for specific days.

```
    public Schedule getSchedule() {
        return schedule;
    }

    public void addLessonToSchedule(DayOfWeek day, String lesson) {
        this.schedule.addLesson(day, lesson);
    }

    public void viewSchedule() {
        System.out.println("Schedule for " + this.getName() + ":");
        System.out.println(this.schedule);
    }
```

The `Researcher` interface defines methods to manage research activities, such as adding research papers, calculating H-Index, and printing research papers.

```
public interface Researcher {
    void addResearchPaper(ResearchPaper paper);
    List<ResearchPaper> getResearchPapers();
    int calculateHIndex();
    void printPapers(java.util.Comparator<ResearchPaper> comparator);
}
```

The `Employee` class is an abstract extension of the `User` class, representing employees in the system. It manages employee-specific tasks, such as message handling.

```
public abstract class Employee extends User {
    private final List<Message> receivedMessages;
    private final List<Message> sentMessages;
    private final Database db;

    public Employee() {
        this(null, null, null, null);
```

The `Teacher` class extends the `Employee` class and implements the `Researcher` interface. It provides functionalities for teaching, research, and complaint management.

```
import java.util.Objects;

public class Teacher extends Employee implements Researcher {
    private final boolean isProfessor;
    private boolean isResearcher;
    private final List<ResearchPaper> researchPapers;
    private static final Database db = Database.getInstance();

```

3. Documentation

Documentation for Code Modules

1. Package: `database`

Class: `DataInitializer`

Initializes the database with sample data for testing and development purposes.

Key Methods:

- `initialize(Database db)` : Populates the database with initial data, such as users and courses.

2. Package: `main.menu`

Class: `AbstractMenu`

Provides a base class for creating menu-driven interfaces with predefined actions and titles.

Key Methods:

- `initializeActions()` : Initializes the menu's action map with options.
- `getMenuTitle()` : Returns the title of the menu.
- `execute()` : Displays the menu and processes user input to execute actions.

Class: `StudentMenu`

Implements a menu for student users to interact with the system, including viewing marks, registering for courses, and managing schedules.

Key Methods:

- `viewMarks()` : Displays the student's marks.
- `registerForCourse()` : Allows the student to register for a course.
- `viewSchedule()` : Shows the student's weekly schedule.

3. Package: `users`

Interface: `Researcher`

Defines methods for managing research papers, calculating H-Index, and printing papers for researchers.

Key Methods:

- `addResearchPaper(ResearchPaper paper)` : Adds a research paper to the researcher's list.
- `getResearchPapers()` : Returns a list of research papers.
- `calculateHIndex()` : Calculates the H-Index based on citations.

Class: `User`

An abstract base class for all users in the system, including students and employees.

Key Methods:

- `checkPassword(String password)` : Validates the user's password.
- `updateNews()` : Notifies the user about new updates.

Class: `Student`

Represents a student with functionalities to manage courses, marks, and schedules

Class: `Student`

Represents a student with functionalities to manage courses, marks, and schedules.

Key Methods:

- `registerForCourse(Course course)` : Registers the student for a course.
- `addLessonToSchedule(DayOfWeek day, String lesson)` : Adds a lesson to the student's schedule.
- `viewSchedule()` : Displays the student's schedule.
- `addMark(Mark mark)` : Adds a mark to the student's record.
- `getMarks()` : Retrieves all marks for the student.
- `getTranscript()` : Returns the student's transcript, including GPA and retake requirements.

Class: `PhDStudent`

Represents a PhD student with additional dissertation-related functionalities.

Key Methods:

- `getDissertationTopic()` : Returns the dissertation topic of the PhD student.
- `getAdvisorName()` : Returns the advisor's name for the dissertation.
- `defendDissertation()` : Simulates the defense of the dissertation.

Class: `MasterStudent`

Represents a master's student with thesis-related functionalities.

Key Methods:

- `getThesisTopic()` : Returns the thesis topic of the master student.

Key Methods:

- `getThesisTopic()` : Returns the thesis topic of the master student.
- `submitThesis(Database db)` : Submits the master's thesis to the database.

4. Package: `utilities`

Class: `Schedule`

Manages the weekly schedule for students and employees.

Key Methods:

- `addLesson(DayOfWeek day, String lesson)` : Adds a lesson to the specified day.
- `removeLesson(DayOfWeek day, String lesson)` : Removes a lesson from the specified day.
- `getLessons(DayOfWeek day)` : Returns the lessons for a specified day.
- `getAllLessons()` : Retrieves all lessons for the entire week.

Class: `Transcript`

Generates a transcript for students, including GPA calculation and course details.

Key Methods:

- `calculateGPA()` : Calculates the GPA based on marks.
- `toString()` : Returns a formatted transcript with course details and GPA.

Class: `Course`

Represents a course with details such as lessons and descriptions.

- `toString()` : Returns a formatted transcript with course details and GPA.

Class: `Course`

Represents a course with details such as lessons and descriptions.

Key Methods:

- `addLesson(Lesson lesson)` : Adds a lesson to the course.
- `getLessons()` : Retrieves the list of lessons in the course.

Class: `Lesson`

Represents a lesson with attributes like name, type, day, and time.

Key Methods:

- `getName()` : Returns the name of the lesson.
- `getType()` : Retrieves the type of the lesson (e.g., Lecture, Practice).
- `getDay()` : Returns the day the lesson is scheduled.
- `getTime()` : Retrieves the time of the lesson.

Class: `Mark`

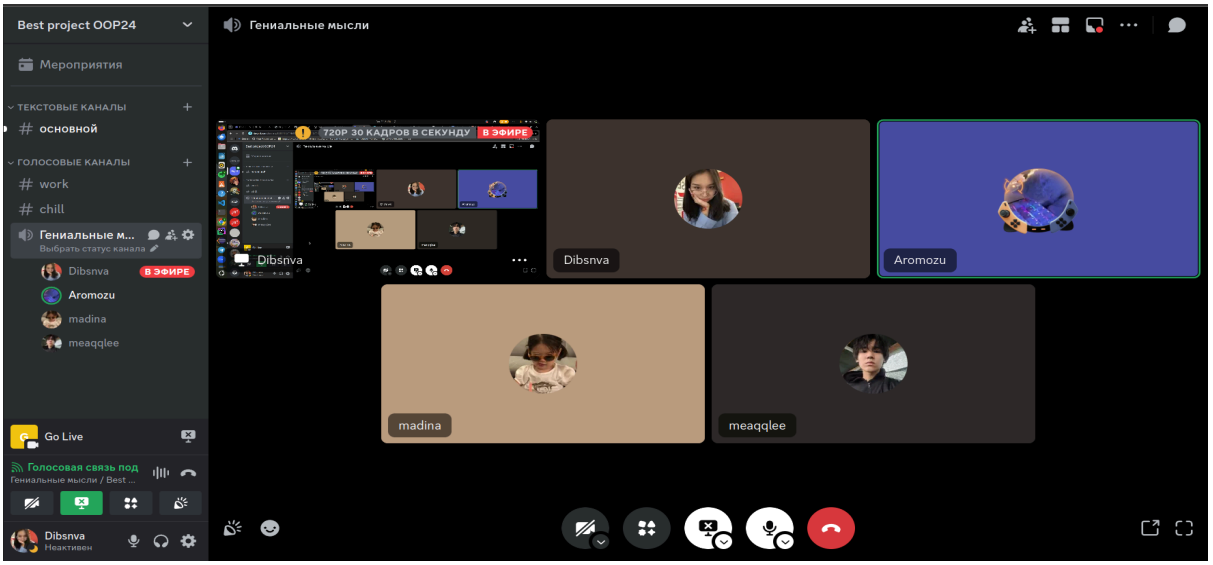
Represents a student's mark for a course.

Key Methods:

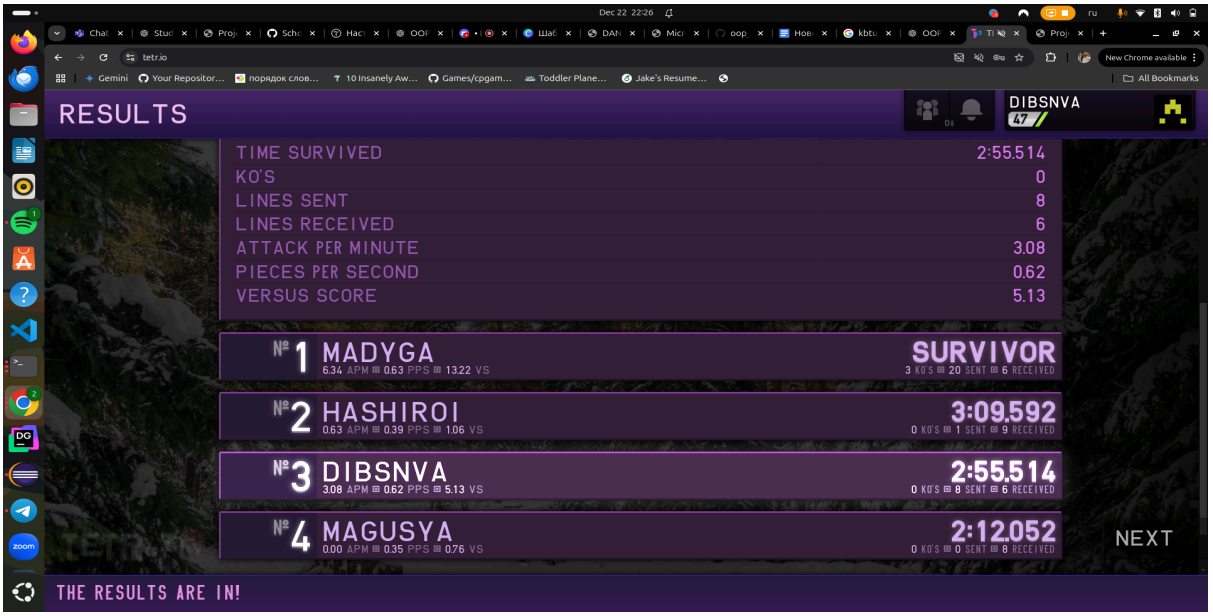
- `getMaxScore(String markType)` : Returns the maximum score possible for a mark type.
- `getScore()` : Retrieves the score for the mark.

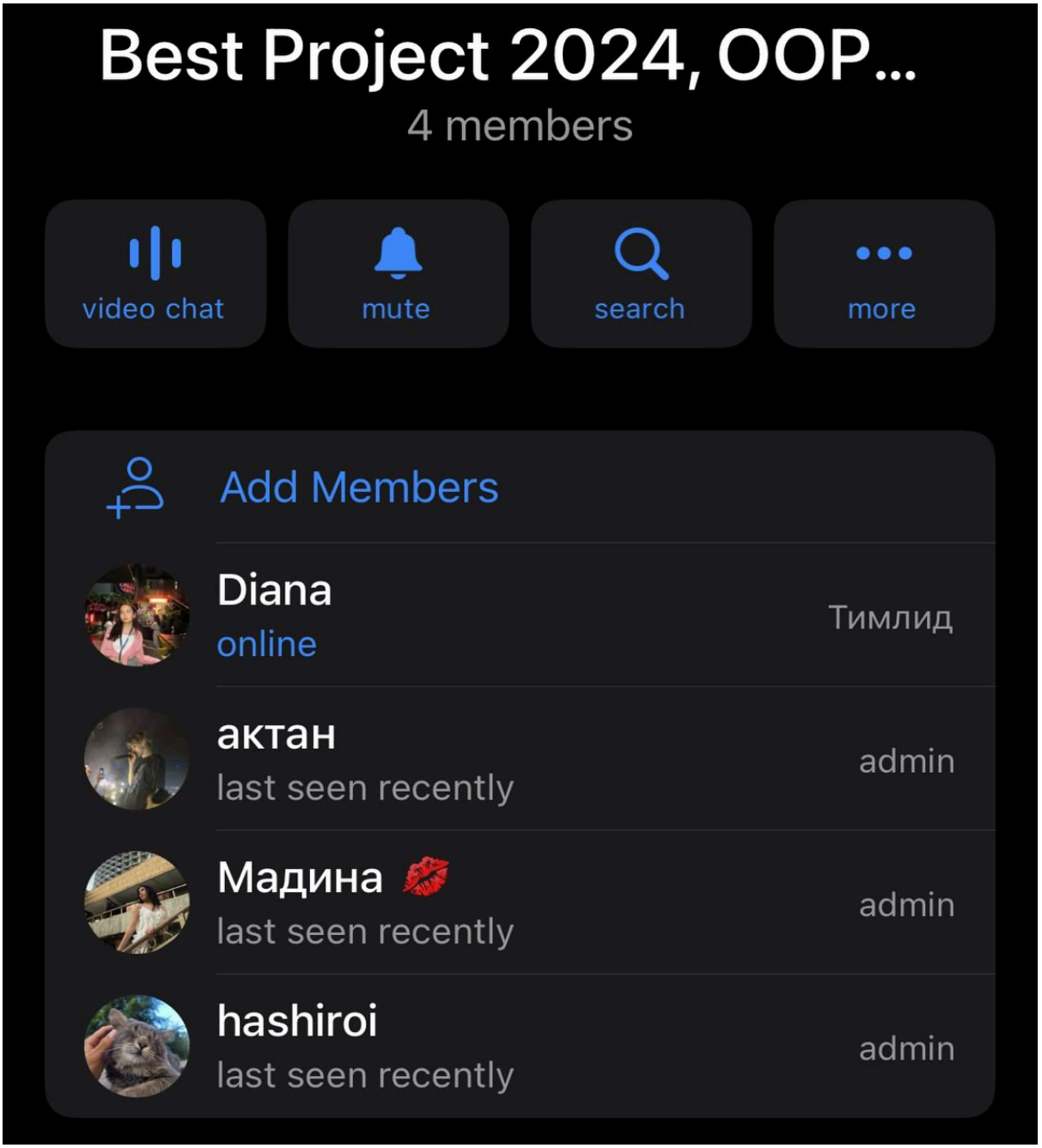
Project Management:

From the very beginning, working together was an absolute delight as we quickly found a shared understanding and camaraderie. We set up a Telegram chat and created a Discord channel, where we would meet almost every evening to collaborate on the project. Our teamwork was seamless, with no conflicts—decisions were made together, and everyone's input was valued. Over time, we became more than just colleagues working on a project; we became friends, sharing not only our tasks but also our interests, news, and knowledge. This bond made our journey even more enjoyable and rewarding.



Even played tetris with each other:)





Problems

- We thought we would not have time to realize everything in a short time, but we did what we wanted and created additional features.